APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **METHOD AND APPARATUS FOR CACHING PREDICATES IN A TRACING FRAMEWORK**

APPLICANT: **Bryan M. CANTRILL**

"EXPRESS MAIL" Mailing Label Number: EL974017436US
Date of Deposit: November 14, 2003

METHOD AND APPARATUS FOR CACHING PREDICATES IN A TRACING FRAMEWORK

Background

[0001] A tracing framework is used to understand the behavior of complex computer systems. Tracing performed using the tracing framework involves recording data from a location or relating to an occurrence in the software of the computer system. In a tracing framework that offers comprehensive coverage, the framework provides a mechanism to allow events *not* to be traced or the user may be flooded with unwanted data.

[0002] Tracing frameworks use predicates to limit the events traced by only tracing data if a certain condition is found to be true. Predicates are a useful mechanism when the user knows which events are interesting. For example, if the user is only interested in an activity associated with a certain process or a certain file descriptor, the user can define a predicate that obtains knowledge about only that particular process or file descriptor.

[0003] Additionally, in a tracing framework that gives a high degree of latitude, evaluating a predicate may have a non-zero cost. Further, if a large number of predicates require evaluation, the cost of evaluating the predicates becomes substantial.

Summary of Invention

[0004] In general, in one aspect, an embodiment of the invention relates to a method for caching in a tracing framework, comprising firing a probe associated with a thread, evaluating a first predicate of the probe, caching the first predicate in a predicate cache associated with the thread, based on the evaluating of the first

predicate and cacheability of the first predicate, and transferring control to the thread, based on the caching.

[0005] In general, in one aspect, an embodiment of the invention relates to a computer system for caching in a tracing framework comprising a processor, a memory, a storage device, and software instructions stored in the memory for enabling the computer system to fire a probe associated with a thread, evaluate a first predicate of the probe, cache the first predicate in a predicate cache associated with the thread, based on the evaluating of the first predicate and cacheability of the first predicate, and transfer control to the thread, based on the caching.

[0006] Other aspects of the invention will be apparent from the following description and the appended claims.

Brief Description of Drawings

[0007] Figure 1 shows a tracing framework in accordance with one or more embodiments of the present invention.

[0008] Figure 2 shows predicate caches in a tracing framework in accordance with one or more embodiments of the present invention.

[0009] Figure 3 shows a method for caching predicates in accordance with one or more embodiments of the present invention.

[0010] Figure 4 shows a networked computer system in accordance with one embodiment of the invention.

Detailed Description

[0011] Specific embodiments of the invention will now be described in detail with reference to the accompanying figures. Like elements in the various figures are denoted by like reference numerals for consistency.

[0012] In the following detailed description of embodiments of the invention, numerous specific details are set forth in order to provide a more thorough understanding of the invention. However, it will be apparent to one of ordinary skill in the art that the invention may be practiced without these specific details. In other instances, well-known features have not been described in detail to avoid obscuring the invention.

[0013] One or more embodiments of the present invention relates to a tracing framework and a method for caching predicates. Figure 1 shows a tracing framework in accordance with one or more embodiments of the present invention. Figure 1 shows source code (100), which defines performance-related questions with respect to a software system. Performance-related questions, for example, may relate to processing speed, resource consumption, and/or proper execution, *etc.*

[0014] In an embodiment of the invention, the source code (100) is obtained using a command line or a graphical user interface. Once obtained, the source code (100) is compiled into executable object code (102). Requests from the object code (102) are communicated to an execution-level tracing framework (108) via an execution interface (106). A tracing framework (108) interprets the requests from the object code (102) and forwards the requests to probe providers (110), which activate certain probes (112 and 114) in an instrumented program (116).

[0015] The probes (112 and 114) correspond to a particular location and/or activity within the instrumented program (116), and answer a specific performance-related question. Further, these probes (112 and 114) may gather the specified information and store the information accordingly. In one or more embodiments, the probe may also have an identifier (id), which may be cached to avoid unnecessary evaluation.

[0016] In one or more embodiments, the probes (112 and 114) may be described in the following psuedo-code:

Code Sample 1

```
1  probe description
2  /predicate/
   {
3  Action
   }
```

[0017] In line 1 of the Code Sample 1, the name of a certain probe is defined. Line 2 shows a predicate, which is a pre-determined, conditional statement that determines whether the action (shown at line 3) of the probe is to be executed. The predicate evaluates to a value of true or false, *e.g.*, an integer value of zero or one, or a defined pointer type. One skilled in the art will appreciate that the Boolean values may be reversed.

[0018] In one or more embodiments of the present invention, a predicate may be assigned an identifier, for example, a predicate cache identifier (id), which may later be stored with its corresponding probe as a probe cache identifier. In one or more embodiments, the predicate cache id is a thirty-two bit, non-zero integer. One skilled in the art will appreciate that a probe may contain more than one predicate/action pairs. Additionally, one skilled in the art will appreciate that two or more predicates of a probe may be identical.

[0019] Line 3 of Code Sample 1 defines the action of the probe that is executed, *i.e.*, the tracing operation. Examples of tracing operations include tracing (or recording data), modifying a state variable, *etc.*

[0020] In one or more embodiments, the present invention caches the predicates in the tracing framework. Generally, caching involves temporarily storing frequently used data to ensure that the data is easily accessible. Caching predicates allows a

predicate (or specifically, an evaluation of a predicate) to be temporarily stored and easily referenced.

[0021] A predicate, which is known to be false, may be identified and stored in a predicate cache. One skilled in the art will understand that storing a predicate that evaluates to true, is typically undesirable, because the cost of the subsequently executed actions amortizes the cost of caching the predicates.

[0022] Figure 2 shows a tracing framework with predicate caches in accordance with one or more embodiments of the present invention. In Figure 2, the tracing framework (108) is associated with a processor (200), which executes an instrumented program (*e.g.*, 116 in Figure 1). Threads (202 A - 202 N) are tasks, which are a sub-set of instructions in the instrumented program (116), executing on processor (200). Predicate caches (204 A - 204 N) are associated with the threads (202 A - 202 N), respectively, and store the predicate of the last cached probe encountered by the particular thread. In particular, the predicate cache id is stored in the predicate cache.

[0023] In one or more embodiments of the present invention, only particular predicates may be cached. For example, a predicate of a probe is defined as cacheable if it references (1) an immutable variable, *i.e.*, a variable that does not change or (2) a thread-specific variable, *i.e.*, a variable with a thread-local state. For example, a thread specific variable may include a thread-specific global variable or a thread-specific dynamically allocated variable. If the predicate is cacheable, the predicate is assigned a unique, non-zero identifier, *i.e.*, a predicate cache id. Further, this predicate cache id is stored with the probe as the probe cache identifier. If a cacheable predicate evaluates to false, the predicate's cache identifier is stored in the evaluating thread's predicate cache.

[0024] Additionally, in one or more embodiments, more than one predicate may be associated with a single probe. A probe having multiple predicates is defined as

cacheable if all the predicates have the same identifier, *i.e.*, the same predicate, which references (1) an immutable variable, or (2) a thread-specific variable. If the predicate is cacheable, the predicate is assigned a unique, non-zero identifier, *i.e.*, a predicate cache id. Similarly, this predicate cache id is stored with the probe as the probe cache identifier. If a cacheable predicate evaluates to false, the predicate's cache identifier is stored in the evaluating thread's predicate cache.

[0025] One skilled in the art will also appreciate that predicate caches may be associated per thread and may be of arbitrary size. Further, predicate caches may cache multiple predicate cache identifiers simultaneously.

[0026] Figure 3 shows caching predicates during thread execution in accordance with one or more embodiments of the present invention. During thread execution, a probe is encountered in the instruction set and the probe fires (Step 300). At this point, a determination is made whether the predicate of the probe is cached and whether the predicate cache is valid (Step 302). Determining whether the predicate is cached may be accomplished in a variety of ways, for example, checking for equivalency of the probe cache id and the predicate cache id stored in the predicate cache. Additionally, determining whether a predicate cache is valid may be accomplished in a variety of ways, for example, ensuring that the probe cache id and the predicate cache id stored in the predicate cache are both non-zero.

[0027] If the predicate of the probe is cached and the predicate cache is valid, then control is transferred back to the thread (Step 304). Otherwise, the predicate of the probe is evaluated (Step 306). If the predicate is cacheable and false (*i.e.*, assuming there is only one predicate in the probe) (Step 308), then the unique, non-zero identifier assigned *a priori*, *i.e.*, a predicate cache id, is stored in the predicate cache associated with that particular thread (Step 310). Additionally, the predicate may already be identified with a predicate cache id, however, previously evaluated to true. In this case, the previously assigned predicate cache id is stored

with the probe and the predicate cache. The probe transfers control to the thread (Step 304). Otherwise, if the predicate is true, then the action of the probe is executed (Step 312) and the probe transfers control to the thread (Step 304).

[0028] In one or more embodiments of the present invention, a predicate cache is initially set to zero and updated as described above. Once a predicate, or specifically, a predicate cache id, is associated with the predicate cache, coherency of the predicate cache must be maintained. Coherency relates to maintaining veritable content in the predicate cache, *i.e.*, ensuring that a thread-specific variable referenced in one or more of the predicates has not changed state, thereby rendering the predicate true.

[0029] Therefore, at any point during thread execution, if a thread-specific variable is stored, the predicate cache is invalidated by assigning the predicate cache id associated with the predicate cache to zero. Further, if a predicate cache is shared by several probes and, during thread execution, one of the probes is no longer cacheable, then the predicate cache is also invalidated by assigning the predicate cache id associated with the predicate cache to zero.

[0030] The invention may be implemented on virtually any type of computer regardless of the platform being used. For example, as shown in Figure 4, a networked computer system (400) includes a processor (402), associated memory (404), a storage device (406), and numerous other elements and functionalities typical of today's computers (not shown). The networked computer (400) may also include input means, such as a keyboard (408) and a mouse (410), and output means, such as a monitor (412). The networked computer system (400) is connected to a local area network (LAN) or a wide area network (*e.g.*, the Internet) via a network interface connection (not shown). Those skilled in the art will appreciate that these input and output means may take other forms. Those skilled in the art will appreciate that one or more elements of the aforementioned

computer (400) may be located at a remote location and connected to the other elements over a network

[0031] In one or more embodiments, the present invention allows a predicate to be evaluated without having to fire a probe, thereby reducing costs associated with probe firing. This allows predicates to be used more efficiently in a tracing framework. The present invention provides a mechanism for caching predicates, while maintaining a coherent cache.

[0032] While the invention has been described with respect to a limited number of embodiments, those skilled in the art, having benefit of this disclosure, will appreciate that other embodiments can be devised which do not depart from the scope of the invention as disclosed herein. Accordingly, the scope of the invention should be limited only by the attached claims.